Automatic Annotation of the Penn-Treebank with LFG F-Structure Information

Aoife Cahill, Mairead McCarthy, Josef van Genabith, Andy Way

School of Computer Applications, Dublin City University
Dublin 9, Ireland
{acahill, mccarthy, josef, away}@computing.dcu.ie

Abstract

Lexical-Functional Grammar f-structures are abstract syntactic representations approximating basic predicate-argument structure. Treebanks annotated with f-structure information are required as training resources for stochastic versions of unification and constraint-based grammars and for the automatic extraction of such resources. In a number of papers (Frank, 2000; Sadler, van Genabith and Way, 2000) have developed methods for automatically annotating treebank resources with f-structure information. However, to date, these methods have only been applied to treebank fragments of the order of a few hundred trees. In the present paper we present a new method that scales and has been applied to a complete treebank, in our case the WSJ section of Penn-II (Marcus et al, 1994), with more than 1,000,000 words in about 50,000 sentences.

1. Introduction

Lexical-Functional Grammar f-structures (Kaplan and Bresnan, 1982; Bresnan, 2001) are abstract syntactic representations approximating basic predicate-argument structure (van Genabith and Crouch, 1996). Treebanks annotated with f-structure information are required as training resources for stochastic versions of unification and constraint-based grammars and for the automatic extraction of such resources. In two companion papers (Frank, 2000; Sadler, van Genabith and Way, 2000) have developed methods for automatically annotating treebank resources with f-structure information. However, to date, these methods have only been applied to treebank fragments of the order of a few hundred trees. In the present paper we present a new method that scales and has been applied to a complete treebank, in our case the WSJ section of Penn-II (Marcus et al, 1994), with more than 1,000,000 words in about 50,000 sentences.

We first give a brief review of Lexical-Functional Grammar. We next review previous work and present three architectures for automatic annotation of treebank resources with f-structure information. We then introduce our new f-structure annotation algorithm and apply it to the Penn-II treebank resource. Finally we conclude and outline further work.

2. Lexical-Functional Grammar

Lexical-Functional Grammar (LFG) is an early member of the family of unification- (more correctly: constraint-) based grammar formalisms (FUG, PATR-II, GPSG, HPSG etc.). It enjoys continued popularity in theoretical and computational linguistics and natural language processing applications and research. At its most basic, an LFG involves two levels of representation: c-structure (constituent structure) and f-structure (functional structure). C-structure represents surface grammatical configurations such as word order and the grouping of linguistic units into larger phrases. The c-structure component of an LFG is represented by a CF-PSG (context-free phrase structure grammar). F-structure represents abstract syntactic functions such as subject, object, predicate etc. in terms of recursive attribute-value structure representations. These abstract syntactic representations abstract away from particulars of surface configuration. The motivation is that while languages differ with respect to surface representation they may still encode the same (or very similar) abstract syntactic functions (or predicate argument structure). To give a simple example, typologically, English is classified as an SVO (subject-verb-object) language while Irish is a verb initial VSO language. However, a sentence like *John saw Mary* and its Irish translation *Chonaic Seán Máire*, while associated with very different c-structure trees, have structurally isomorphic f-structure representations, as represented in **Figure 1**.

C-structure trees and f-structures are related in terms of projections (indicated by the arrows in the examples in **Figure 1**). These projections are defined in terms of f-structure annotations in c-structure trees (describing f-structures) originating from annotated grammar rules and lexical entries. A sample set of LFG grammar rules with functional annotations (f-descriptions) is provided in **Figure 2**. Optional constituents are indicated by brackets.

3. Previous Work: Automatic Annotation Architectures

It would be desirable to have a treebank annotated with f-structure information as a training resource for probabilistic constraint (unification) grammars and as a resource for extracting such grammars. The large number of CFG rule types in treebanks ($> 19,000$ for Penn-II) makes manual f-structure annotation of grammar rules extracted from complete treebanks prohibitively time consuming and expensive. Recently, in two companion papers (Frank, 2000; Sadler, van Genabith and Way, 2000) have investigated the possibility of automatically annotating treebank resources with f-structure information. As far as we are aware, we can distinguish three different types of automatic f-structure annotation architectures (these have all been developed within an LFG framework and although we refer to these as automatic f-structure annotation architectures they

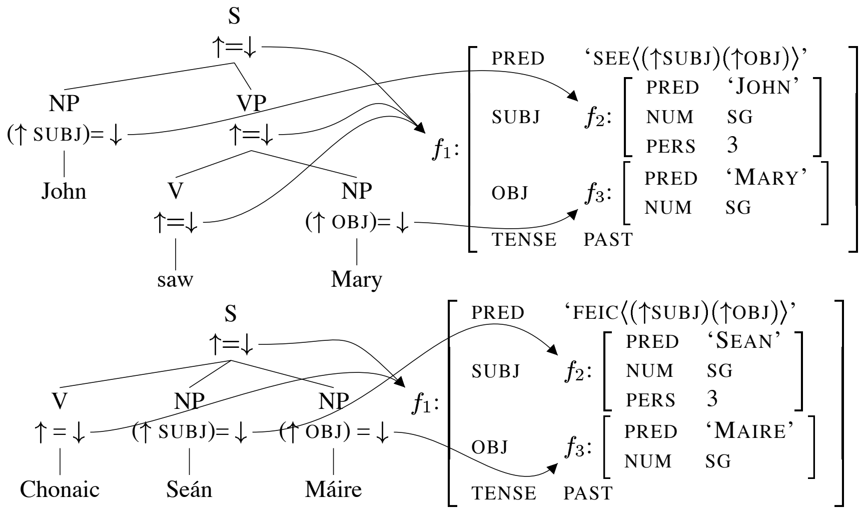

S
↑=↓
NP
(↑ SUBJ)= ↓
John
VP
↑=↓
V
↑=↓
saw
NP
(↑ OBJ)= ↓
Mary
f_1:
PRED ‘SEE⟨(↑SUBJ)(↑OBJ)⟩’
SUBJ f_2: PRED ‘JOHN’ NUM SG PERS 3
OBJ f_3: PRED ‘MARY’ NUM SG
TENSE PAST
S
↑=↓
V
↑ = ↓
Chonaic
NP
(↑ SUBJ)= ↓
Seán
NP
(↑ OBJ) = ↓
Máire
f_1:
PRED ‘FEIC⟨(↑SUBJ)(↑OBJ)⟩’
SUBJ f_2: PRED ‘SEAN’ NUM SG PERS 3
OBJ f_3: PRED ‘MAIRE’ NUM SG
TENSE PAST

Figure 1: C- and f-structures for an English and corresponding Irish sentence

$$
\begin{array}{lcccccc}
\text{S} & \rightarrow & \begin{array}{c}\text{NP}\\ \uparrow \text{SUBJ} =\downarrow\end{array} & \begin{array}{c}\text{VP}\\ \uparrow=\downarrow\end{array} & \left(\begin{array}{c}\text{ADV}\\ \downarrow\in\uparrow \text{ADJN}\end{array}\right) & & \\
\\
\text{NP} & \rightarrow & \begin{array}{c}\text{Det}\\ \uparrow=\downarrow\end{array} & \begin{array}{c}\text{N}\\ \uparrow=\downarrow\end{array} & & & \\
\\
\text{VP} & \rightarrow & \begin{array}{c}\text{V}\\ \uparrow=\downarrow\end{array} & \left(\begin{array}{c}\text{NP}\\ \uparrow \text{OBJ} =\downarrow\end{array}\right) & \left(\begin{array}{c}\text{VP}\\ \uparrow \text{XCOMP} =\downarrow\end{array}\right) & \left(\begin{array}{c}\text{S}\\ \uparrow \text{COMP} =\downarrow\end{array}\right) &
\end{array}
$$

Figure 2: Sample LFG grammar rules for a fragment of English

could equally well be used to annotate treebanks with e.g. HPSG feature structure or with Quasi-Logical Form (QLF) (Liakata and Pulman, 2002) annotations):

- regular expression based annotation (Sadler, van Genabith and Way, 2000)
- tree description set-based rewriting (Frank, 2000)
- annotation algorithms

More recently, we have learnt about the QLF annotation work by (Liakata and Pulman, 2002). Much like (Frank, 2000), their approach is based on matching configurations in a flat, set-based tree description representation.

Below we will briefly describe the first two architectures. The new work presented in this paper is based on an annotation algorithm and discussed at length in Sections 4 and 5 of the paper.

3.1. Regular Expression-Based Annotation

(Sadler, van Genabith and Way, 2000) describe a regular expression based automatic f-structure annotation methodology. The basic idea is very simple: first, the CFG rule set is extracted from the treebank (fragment); second, regular-expression based annotation principles are defined; third, the principles are automatically applied to the rule set to generate an annotated rule set; fourth, the annotated rules are automatically matched against the original treebank trees and thereby f-structures are generated for these trees. Since the annotation principles factor out linguistic generalisations, their number is much smaller than the number of CFG treebank rules. In fact, the regular expression-based f-structure annotation principles constitute a principle-based LFG c-structure/f-structure interface. We will explain the method in terms of a simple example. Let us assume that from the treebank trees we extract CFG rules expanding `vp` of the form (amongst others):

```
vp:A > v:B s:C
vp:A > v:B v:C s:D
vp:A > v:B v:C v:D s:E
     ..
vp:A > v:B s:C pp:D
vp:A > v:B v:C s:D pp:E
vp:A > v:B v:C v:D s:E pp:F
     ..
vp:A > advp:B v:C s:D
vp:A > advp:B v:C v:D s:E
vp:A > advp:B v:C v:D v:E s:F
     ..
vp:A > advp:B v:C s:D pp:E
vp:A > advp:B v:C v:D s:E pp:F
vp:A > advp:B v:C v:D v:E s:F pp:G
```

Each CFG category in the rule set has been associated with a logical variable designed to carry f-structure information. In order to annotate these rules we can define a set of regular-expression based annotation principles:

```
vp:A > * v:B v:C *
```

```
          @ [B:xcomp=C,B:subj=C:subj]
    vp:A > *(˜v) v:B *
          @ [A=B]
    vp:A > * v:B s:C *
          @ [B:comp=C]
```

The first annotation principle states that if anywhere in a rule RHS expanding a vp category we find a v v sequence the f-structure associated with the second v is the value of an xcomp attribute in the f-structure associated in the first v ('*' is the Kleene star and, if unattached to any other regular expression, signifies any string). It is easy to see how this annotation principle matches many of the extracted example rules, some even twice. The second principle states that the leftmost v in vp rules is the head. The leftmost constraint is expressed by the fact that the rule RHS may consist of an initial string that may not contain a v: *(˜v). Each of the annotation principles is partial and underspecified: they underspecify CFG rule RHSs and annotate matching rules partially. The annotation interpreter applies all annotation principles to each CFG rule as often as possible and collects all resulting annotations. It is easy to see that we get, e.g., the following (partial) annotation for:

```
    vp:A > advp:B v:C v:D v:E s:F pp:G
          @ [A=C,
             C:xcomp=D,C:subj=D:subj,
             D:xcomp=E,D:subj=E:subj,
             E:comp=F]
```

In their experiments with the publicly available subsection of the AP treebank, (Sadler, van Genabith and Way, 2000) achieve precision and recall results in the low to mid 90 percent region against a "gold standard" manually annotated grammar. The method is order independent, partial and robust. To date, however, the method has been applied to only small CFG rule sets (of the order of 500 rules approx.).

3.2. Rewriting of Flat Tree Description Set Representations

In a companion paper, (Frank, 2000) develops an automatic annotation method that in many ways is a generalisation of the regular expression-based annotation method. The basic idea is again simple: first, trees in treebanks are translated into a flat set representation format in a tree description language; second, annotation principles are defined in terms of rewriting rules employing a rewriting system originally developed for transfer based machine translation architectures (Kay, 1999). We will illustrate the method with a simple example

```
       s:A
      /   \              dom(A,B), dom(A,C),
    np:B   vp:C          dom(C,D), ..
     |      |      =>    pre(B,C),
    John    v:D          cat(A,s), cat(C,vp),
            |            cat(D,v), ..
           left

       dom(X,Y), dom(X,Z), pre(Y,Z),
       cat(X,s), cat(Y,np), cat(Z,vp)
           ==>
       subj(X,Y), eq(X,Z)
```

Trees are described in terms of (immediate and general) dominance and precedence relations, labelling functions assigning categories to nodes and so forth. In our example, node identifiers A, B, etc. do double duty as f-structure variables. The annotation principle states that if node X dominates both Y and Z and if Y preceds Z and the respective CFG categories are s, np and vp, then Y is the subject of X and Z is the same as (i.e. is the head of) X.

The tree description rewriting method has a number of advantages:

- in contrast to the regular expression-based method, annotation principles formulated in the flat tree description method can consider arbitrary tree fragments (and not just only local CFG rule configurations).

- in contrast to the regular expression based method which is order independent, the rewriting technology can be used to formulate both order-dependent and order-independent systems. Cascaded, order-dependent systems can support a more compact and perspicuous statement of annotation principles as certain transformations can be assumed to have already applied earlier on in the cascade.

For a more detailed, joint presentation of the two approaches consult (Frank et al, 2002). Like the regular expression based annotation method, the tree description based set rewriting method has to date only been applied to small treebank fragments of the order of several hundred trees.

3.3. Annotation Algorithms

The previous two automatic annotation architectures enforce a clear separation between the statement of annotation principles and the annotation procedure. In the first case, the annotation procedure is provided by our regular expression interpreter, in the second by the set rewriting machinery. A clean separation between principles and processing supports maintenance and reuse of annotation principles. There is, however, a third possible automatic annotation architecture, namely an annotation algorithm. In principle, two variants are possible. An annotation algorithm may

- directly (recursively) transduce a treebank tree into an f-structure – such an algorithm would more appropriately be referred to as a tree to f-structure transduction algorithm;

- annotate CFG treebank trees with f-structure annotations from which an f-structure can be computed by a constraint solver.

The first mention of an automatic f-structure annotation algorithm we are aware of is unpublished work by Ron Kaplan (p.c.) who as early as 1996 worked on automatically generating f-structures from the ATIS corpus to generate data for LFG-DOP (Bod and Kaplan, 1998) applications. Kaplan's approach implements a direct tree to f-structure transduction. The algorithm walks the tree looking for different configurations (e.g. np under s, 2nd np under vp, etc.) and "folds" the tree into the corresponding f-structure. In contrast, our approach develops the second, more indirect tree annotation algorithm paradigm. We have designed and implemented an algorithm that annotates nodes in the Penn-II treebank trees with f-structure constraints. The design and the application of the algorithm is explained below.

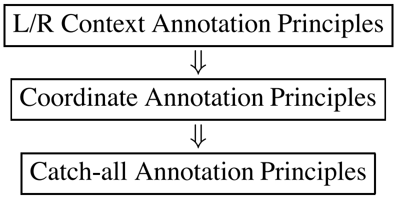
4. Automatic Annotation Algorithm Design

In our work on the automatic annotation algorithm we an annotation method that is robust and scales to the whole of the Penn-II treebank with 19,000 CFG rules for 1,000,000 words with approx. 50,000 sentences. The algorithm is implemented as a recursive procedure (in Java) which annotates Penn-II treebank tree nodes with f-structure information. The annotations describe what we call "proto-f-structures", which

- encode basic predicate-argument-modifier structures;
- may be partial or unconnected (i.e. in some cases a sentence may be associated with two or more unconnected f-structure fragments rather than a single f-structure);
- may not encode some reentrancies, e.g. in the case of wh- and other movement or distribution phenomena (of subjects into VP coordinate structures etc.).

Compared to the regular expression- and the set rewriting-based annotation methods described above, the new algorithm is somewhat more coarse-grained, both with respect to resulting f-structures and with respect to the formulation of the annotation principles.

Even though the method is encoded in the form of an annotation algorithm (i.e. a procedure), we did not want to completely hard code the linguistic basis for the annotation into the procedure. In order to achieve a clean design which supports maintainability and reusability of the annotation algorithm and the linguistic information encoded in it, we decided to design the algorithm in terms of three main components that work in sequence:


L/R Context Annotation Principles
⇓
Coordinate Annotation Principles
⇓
Catch-all Annotation Principles

Each of the components of the algorithm is presented below.

In addition, at the lexical level, for each Penn-II preterminal category type, we have a lexical macro associating any terminal under the category with the required f-structure information. To give a simple example, a singular common noun nns, such as e.g. *company*, is annotated by the lexical macro for nns as $\uparrow \texttt{pred} = \texttt{company}$, $\uparrow \texttt{num} = \texttt{sg}$, $\uparrow \texttt{pers} = \texttt{3rd}$.

4.1. L/R Context Annotation Principles

The annotation algorithm recursively traverses trees in a top-down fashion. Apart from very few exceptions (e.g. possessive NPs), at each stage of the recursion the algorithm considers local subtrees of depth one (i.e. effectively CFG rules). Annotation is driven by categorial and simple configurational information in a local subtree.

In order to annotate the nodes in the trees, we partition each sequence of daughters in a local subtree (i.e. rule RHS) into three sections: left context, head and right context. The head of a local tree is computed using Collins (1999) head lexicalised grammar annotation scheme (except for coordinate structures, where we depart from Collins' head scheme). In a preprocessing step we transform the treebank into head lexicalised form. During automatic annotation, we can then easily identify the head constituent in a local tree as that constituent which carries the same terminal string as the mother of the local tree. With this we can compute left and right context: given the head constituent, the left context is the prefix of the local daughter sequence while the right context is the suffix. For each local tree we also keep track of the mother category. In addition to the positional (reduced to the simple tripartition into head with left/right context) and categorial information about mother and daughter nodes, we also employ an LFG distinction between subcategorisable (subj, obj, obj2, obl, xcomp, comp ...) and non-subcategorisable (adjn, xadjn ...) grammatical functions. Subcategorisable grammatical functions characterise arguments, while non-subcategorisable functions characterise adjuncts (modifiers).

Using this information we construct what we refer to as an "annotation matrix" for each of the rule LHS categories in the Penn-II treebank grammar. The x-axis of the matrix is given by the tripartition into left context, head and right context. The y-axis is defined by the distinction between subcategorisable and non-subcategorisable grammatical functions.

Consider a much simplified example: for rules (local trees) expanding English np's, the rightmost nominal (n, nn, nns etc.) on the RHS is (usually) the head. Heads are annotated $\uparrow = \downarrow$. Any det or quant constituent in the left context is annotated $\uparrow \texttt{spec} = \downarrow$. Any adjp in the left context is annotated $\downarrow \in \uparrow \texttt{adjn}$. Any nominal in the left context (in noun noun sequences) is annotated as a modifier $\downarrow \in \uparrow \texttt{adjn}$. Any pp in the right context is annotated as $\downarrow \in \uparrow \texttt{adjn}$. Any relcl in the right context as $\downarrow \in \uparrow \texttt{relmod}$, any nominal (phrase - usually separated by commas following the head) as an apposition $\downarrow \in \uparrow \texttt{app}$ and so forth. Information such as this is used to populate the np annotation matrix, partially represented in **Table 1**.

In order to minimise mistakes, the annotation matrices are very conservative: subcategorisable grammatical functions are only assigned if there is no doubt (e.g. an np following a preposition in a pp is assigned $\uparrow \texttt{obj} = \downarrow$; a vp following a v in a vp constituent is assigned $\uparrow \texttt{xcomp} = \downarrow$, $\uparrow \texttt{subj} = \uparrow \texttt{xcomp}:\texttt{subj}$ and so forth). If, for any constituent, the argument - modifier status is in doubt, we annotate the constituent as an adjunct: $\downarrow \in \uparrow \texttt{adjn}$.

Treebanks have an interesting property: for each cate-

np	left context	head	right context
subcat functions	det, quant : ↑ spec =↓	n, nn, nns : ↑=↓	...
non-subcat functions	adjp : ↓∈↑ adjn n, nn, nns : ↓∈↑ adjn ...		relcl : ↓∈↑ relmod pp : ↓∈↑ adjn n, nn, nns : ↓∈↑ app

Table 1: Simplified, partial annotation matrix for np rules

gory, there is a small number of very frequently occurring rules expanding that category, followed by a large number of less frequent rules many of which occur only once or twice in the treebank (Zipf's law).

For each particular category, the corresponding annotation matrix is constructed from the most frequent rules expanding that category. In order to guarantee similar coverage for the annotation matrices for the different rule LHS in the Penn-II treebank, we design each matrix according to an analysis of the most frequent CFG rules expanding that category, such that the token occurrences of those rules cover at least 80% of the token occurrences of all rules expanding that LHS category in the treebank. In order to do this we need to look at the following number of most frequent rule types for each category given in **Table 2**.

Although constructed based on the evidence of the most frequent rule types, the resulting annotation matrices do generalise to as yet unseen rule types in the following two ways:

- during the application of the annotation algorithm, annotation matrices annotate less frequent, unseen rules with constituents matching the left/right context and head specifications. The resulting annotation might be partial (i.e. some constituents in less frequent rule types may be left unannotated).

- in addition to monadic categories, the Penn-II treebank contains versions of these categories associated with functional annotations (-LOC, -TMP etc. indicating locative, temporal, etc. and other functional information). If we include functional annotations in the categories, there are approx. 150 distinct LHS categories in the CFG extracted from the Penn-II treebank resource. Our annotation matrices were developed with the most frequent rule types expanding monadic categories only. During application of the annotation algorithm, the annotation matrix for any given monadic category C is also applied to all rules (local trees) expanding C-LOC, C-TMP etc., i.e. instances of the category carrying functional information.

In our work to date we have not yet covered "constituents" marked frag(ment) and x (unknown constituents) in the Penn-II treebank.

Finally, note that L/R context annotation principles are only applied if the local tree (rule RHS) does not contain any instance of a coordinating conjunction cc. Constructions involving coordinating conjunctions are treated separately in the second component of the annotation algorithm.

4.2. Coorordinating Conjunction Annotation Principles

Coordinating constructions come in two forms: like and unlike (UCPs) constituent coordinations. Due to the (often too) flat treebank analyses, these present special problems. Because of this, an integrated treatment of coordinate structures with the other annotation principles would have been too complex and messy. For this reason we decided to treat coordinate structures in a separate module. Here we only have space to talk about like constituent coordinations.

The annotation algorithm first attempts to establish the head of a coordinate structure (usually the rightmost coordination) and annotates it accordingly. It then uses a variety of heuristics to find and annotate the various coordinated elements. One of the heuristics employed simply states that if both the immediate left and the immediate right constituents next to the coordination have the same category, then find all such categories in the left context of the rule and annotate these together with the immediate left and right constituents of the coordination as individual elements ↓∈↑ coord in the f-structure set representation of the coordination.

4.3. Catch-All Annotation Principles

The final component of the algorithm utilises functional information provided in the Penn-II treebank annotations. Any constituent, no matter what category, left unannotated by the previous two annotation algorithm components, that carries a Penn-II functional annotation other than SBJ and PRD, is annotated as an adjunct ↓∈↑ adjn.

5. Results and Evaluation

The annotation algorithm is implemented in terms of a Java program. Annotation of the complete WSJ section of the Penn-II treebank takes less than 30 minutes on a Pentium IV PC. Once annotated, for each tree we collect the feature structure annotations and feed them into a simple constraint solver implemented in Prolog.

Our constraint solver can handle equality constraints, disjunction and simple set-valued feature constraints. Currently, however, our annotations do not involve disjunctive constraints. This means that for each tree in the treebank we either get a single f-structure, or, in the case of partially annotated trees, a number of unconnected f-structure fragments, or, in case of feature structure clashes, no f-structure.

As pointed out above, in our work to date we have not developed an annotation matrix for frag(mentary) constituents. Furthermore, as it stands, the algorithm completely ignores "movement" (or dislocation and control)

ADJP	ADVP	CONJP	FRAG	LST	NAC	NP	NX	PP	PRN	PRT	QP	RRC
25	3	3	184	4	6	64	14	2	35	2	11	12

S	SBAR	SBARQ	SINV	SQ	UCP	VP	WHADJP	WHADVP	WHNP	WHPP	X	
11	3	20	16	68	78	146	2	2	2	1	37	

Table 2: # of most frequent rule types analysed to construct annotation matrices

phenomena marked in the Penn-II annotations in terms of coindexation (of traces). This means that the f-structures generated in our work to date miss some reentrancies which a more fine-grained analysis would show.

Furthermore, because of the limited capabilities of our constraint solver, in our current work we cannot use functional uncertainty constraints (regular expression based constraints over paths in f-structure) to localise unbounded dependencies to model "movement" phenomena. Also, again because of limitations of our constraint solver, we cannot express subsumption constraints in our annotations to, e.g., distribute subjects into coordinate `vp` structures.

To give an illustration of our method, we give the first sentence of the Penn-II treebank and the f-structure generated as an example in **Figure 3**.

Currently we obtain the following general results with our automatic annotation algorithm summarised in **Table 3**:

# f-structure (fragments)	# sentences	percentage
0	2701	5.576
1	38188	78.836
2	4954	10.227
3	1616	3.336
4	616	1.271
5	197	0.407
6	111	0.229
7	34	0.070
8	12	0.024
9	6	0.012
10	4	0.008
11	1	0.002

Table 3: Automatic annotation results

The Penn-II treebank contains 49167 trees. The results reported in **Table 3** ignore 727 trees containing `frag`(ment) and `x` (unknown) constituents as we did not provide any annotation for them in our work to date. At this early stage of our work, 38188 of the trees are associated with a complete f-structure. For 2701 trees no f-structure is produced (due to feature clashes). 4954 are associated with 2 f-structure fragments, 1616 with 3 fragments and so forth.

5.1. Evaluation

In order to evaluate the results of our automatic annotation we distinguish between "qualitative" and "quantitative" evaluation. Qualitative evaluation involves a "gold-standard", while quantitative evaluation does not.

5.1.1. Qualitative Evaluation

Currently, we evaluate the output generated by our automatic annotation qualitatively by manually inspecting the f-structures generated. In order to automate the process we are currently working on a set of 100 randomly selected sentences from the Penn-II treebank to manually construct gold-standard annotated trees (and hence f-structures). These can then be processed in a number of ways:

- manually annotated gold-standard trees can be compared with the automatically annotated trees using the labelled bracketing precision and recall measures from `evalb`, a standard software package to evaluate PCFG parses. This presupposes that we treat annotated tree nodes as atoms (i.e. a complex string such as `np:`↑ `obj` =↓ is treated as an atomic label) and that in cases where nodes receive more than one f-structure annotation the order of these is the same in both the gold-standard and the automatically annotated version.

- gold-standard and automatically generated f-structures can be translated into a flat set of functional descriptions (`pred(A,see), subj(A,B), pred(B,John), obj(A,C), pred(C,Mary)`) and precision and recall can be computed for those.

- f-structures can be transformed (or unfolded) into trees by sorting attributes alphabetically at each level of embedding and by coding reentrancies as indices. After this transformation, gold-standard and automatically generated f-structures can be compared using `evalb`. This presupposes that both the gold-standard and the automatically generated f-structure have identical "terminal" yield.

5.1.2. Quantitative Evaluation

For purely quantitative evaluation (that is evaluation that does not necessarily assess the quality of the generated resources), we currently employ two related measures. These measures give an indication as to how partial our automatic annotation is at the current stage of the project. The first measure is the percentage of RHS constituents in grammar rules that receive an annotation. The table lists the annotation percentage for RHS elements of some of the Penn-II LHS categories. Because of the functional annotations provided in Penn-II, the complete list of LHS categories would contain approx. 150 entries. Note that the percentages listed below ignore punctuation markers (which are not annotated):

```
Pierre Vinken, 61 years old, will join the board as a nonexecutive
director Nov. 29.

( S ( NP-SBJ ( NP ( NNP Pierre ) ( NNP Vinken ) ) ( , , ) ADJP ( NP (
 CD 61 ) ( NNS years ) ) ( JJ old ) ) ( , , ) ) ( VP ( MD will ) ( VP
 ( VB join ) ( NP ( DT the ) ( NN board ) ) ( PP-CLR ( IN as ) ( NP (
 DT a ) ( JJ nonexecutive ) ( NN director ) ) ) ( NP-TMP ( NNP Nov. )
 CD 29 ) ) ) ) ( . . ) )

subj : headmod : 1 : num : sing
                     pers : 3
                     pred : Pierre
       num : sing
       pers : 3
       pred : Vinken
       adjunct : 2 : adjunct : 3 : adjunct : 4 : pred : 61
                                   pers : 3
                                   pred : years
                                   num : pl
                     pred : old
xcomp : subj : headmod : 1 : num : sing
                             pers : 3
                             pred : Pierre
               num : sing
               pers : 3
               pred : Vinken
               adjunct : 2 : adjunct : 3 : adjunct : 4 : pred : 61
                                           pers : 3
                                           pred : years
                                           num : pl
                             pred : old
        obj : spec : det : pred : the
              num : sing
              pers : 3
              pred : board
        obl : obj : spec : det : pred : a
                    adjunct : 5 : pred : nonexecutive
                    pred : director
                    num : sing
                    pers : 3
              pred : as
        pred : join
        adjunct : 6 : pred : Nov.
                      num : sing
                      pers : 3
                      adjunct : 7 : pred : 29
pred : will
modal : +
```

Figure 3: F-structure generated for the first sentence in Penn-II

LHS	# RHS elements	# RHS annotated	% annotated
ADJP	1653	1468	88.80
ADJP-ADV	21	21	100.00
ADJP-CLR	27	24	88.88
ADV	607	532	87.64
NP	30793	29145	94.64
PP	1090	905	83.02
S	14912	13144	88.14
SBAR	423	331	78.25
SBARQ	270	212	78.51
SQ	657	601	91.47
VP	40990	35693	87.07

The second, related measure gives the average number of f-structure fragments generated for each treebank tree (the more partial our annotation the more unconnected f-structure fragments are generated for a sentence). For 45739 sentences, the average number of fragments per sentences is currently: 1.26 (note again that the number excludes sentences containing `frag` and `x` constituents).

6. Conclusion and Further Work

In this paper we have presented an automatic f-structure annotation algorithm and applied it to annotate the Penn-II treebank resource with f-structure information. The resulting representations are proto-f-structures showing basic predicate-argument-modifier structure. Currently, 38,188 sentences (78.8% of the 48,440 trees without `frag` and `x` constituents) receive a complete f-structure; 4954 sentences are associated with two f-structure fragments, 1,616 with three fragments. 2,701 sentences are not associated with an f-structure.

In future work we plan to extend and refine our automatic annotation algorithm in a number of ways:

- We are working on reducing the amount of f-structure fragmentation by providing more complete annotation principles.

- Currently the `pred` values (i.e. the predicates) in the f-structures generated are surface (i.e. inflected) rather than root forms. We are planning to use the output of a two-level morphology to annotate the Penn-II strings with root forms which can then be picked up by our lexical macros and used as `pred` values in the automatic annotations.

- Currently our annotation algorithm ignores the Penn-II encoding of "moved" constituents in topicalisation, wh-constructions, control constructions and the like. These (often non-local) dependencies are marked in the Penn-II tree annotations in terms of indices. In future work we intend to make our annotation algorithm sensitive to such information. There are two (possibly complementary) ways of achieving this: the first is to make the annotation algorithm sensitive to the index scheme provided by the Penn-II annotations either during application of the algorithm or in terms of undoing "movement" in a treebank preprocessing step. The latter route is explored in recent work by (Liakata and Pulman, 2002). The second possibility is to use the LFG machinery of functional uncertainty equations to effectively localise unbounded dependency relations in a functional annotation at a particular node. Functional uncertainty equations allow the statement of regular expression-based paths in f-structure. Currently we cannot resolve such paths with our constraint solver.

- We are currently experimenting with probabilistic grammars extracted from the automatically annotated version of the Penn-II treebank. We will be reporting on the results of these experiments elsewhere (Cahill et al, 2002).

- We are planning to exploit the f-structure/QLF/UDRS correspondences established by (van Genabith and Crouch, 1996; van Genabith and Crouch, 1997) to generate semantically annotated versions of the Penn-II treebank.

Acknowledgements

This research was part-funded by Enterprise Ireland Basic Research grant SC/2001/186.

7. References

R. Bod and R. Kaplan 1998. A probabilistic corpus-driven model for lexical-functional grammar. In: *Proceedings of Coling/ACL'98*. 145–151.

J. Bresnan 2001. *Lexical-Functional Syntax*. Blackwell, Oxford.

A. Cahill, M. McCarthy, J. van Genabith and A. Way 2002. Parsing with a PCFG Derived from Penn-II with an Automatic F-Structure Annotation Procedure. In: *The sixth International Conference on Lexical-Functional Grammar*, Athens, Greece, 3 July - 5 July 2002 to appear (2002)

M. Collins 1999. *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

A. Frank. 2000. Automatic F-Structure Annotation of Treebank Trees. In: (eds.) M. Butt and T. H. King, *The fifth International Conference on Lexical-Functional Grammar*, The University of California at Berkeley, 19 July - 20 July 2000, CSLI Publications, Stanford, CA.

A. Frank, L. Sadler, J. van Genabith and A. Way 2002. From Treebank Resources to LFG F-Structures. In: (ed.) Anne Abeille, *Treebanks: Building and Using Syntactically Annotated Corpora*, Kluwer Academic Publishers, Dordrecht/Boston/London, to appear (2002)

M. Kay 1999. Chart Translation. In *Proceedings of the Machine Translation Summit VII*. "MT in the great Translation Era". 9–14.

R. Kaplan and J. Bresnan 1982. Lexical-functional grammar: a formal system for grammatical representation. In Bresnan, J., editor 1982, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge Mass. 173–281.

M. Liakata and S. Pulman 2002. *From trees to predicate-argument structures*. Unpublished working paper. Centre for Linguistics and Philology, Oxford University.

M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, M. Ferguson, K. Katz and B. Schasberger 1994. The Penn Treebank: Annotating Predicate Argument Structure. In: *Proceedings of the ARPA Human Language Technology Workshop*.

L. Sadler, J. van Genabith and A. Way. 2000. Automatic F-Structure Annotation from the AP Treebank. In: (eds) M. Butt and T. H. King, *The fifth International Conference on Lexical-Functional Grammar*, The University of California at Berkeley, 19 July - 20 July 2000, CSLI Publications, Stanford, CA.

J. van Genabith and D. Crouch 1996. Direct and Underspecified Interpretations of LFG f-Structures. In: *COLING 96*, Copenhagen, Denmark, Proceedings of the Conference. 262–267.

J. van Genabith and D. Crouch 1997. On Interpreting f-Structures as UDRSs. In: *ACL-EACL-97*, Madrid, Spain, Proceedings of the Conference. 402–409.